**HALL TICKET NUMBER: 2403A51365**

**NAME: RAYAPUDI HARITHA**

**BATCH: 24BTCAICSB14**

**AssignmentNumber:8.1**

**Lab 8: Test-Driven Development with AI – Generating and Working  
with Test Cases  
Lab Objectives:  
• To introduce students to test-driven development (TDD) using  
AI code generation tools.  
Week4 -  
Monday**

**• To enable the generation of test cases before writing code  
implementations.  
• To reinforce the importance of testing, validation, and error  
handling.  
• To encourage writing clean and reliable code based on AI-  
generated test expectations.  
Lab Outcomes (LOs):  
After completing this lab, students will be able to:  
• Use AI tools to write test cases for Python functions and classes.  
• Implement functions based on test cases in a test-first  
development style.  
• Use unittest or pytest to validate code correctness.  
• Analyze the completeness and coverage of AI-generated tests.  
• Compare AI-generated and manually written test cases for quality  
and logic**

**Task Description #1 (Password Strength Validator – Apply AI in  
Security Context)  
• Task: Apply AI to generate at least 3 assert test cases for  
is\_strong\_password(password) and implement the validator  
function.  
• Requirements:  
o Password must have at least 8 characters.  
o Must include uppercase, lowercase, digit, and special  
character.  
o Must not contain spaces.  
Example Assert Test Cases:  
assert is\_strong\_password("Abcd@123") == True  
assert is\_strong\_password("abcd123") == False  
assert is\_strong\_password("ABCD@1234") == True**

**Prompt:-**

• Task: Apply AI to generate at least 3 assert test cases for  
is\_strong\_password(password) and implement the validator  
function.  
• Requirements:  
o Password must have at least 8 characters.  
o Must include uppercase, lowercase, digit, and special  
character.  
o Must not contain spaces.  
Example Assert Test Cases:  
assert is\_strong\_password("Abcd@123") == True  
assert is\_strong\_password("abcd123") == False  
assert is\_strong\_password("ABCD@1234") == True

Code And Output:-

A screenshot of a computer

AI-generated content may be incorrect.

**Code Explanation:-**

The code defines a function named is\_strong\_password designed to evaluate the strength of a given password string. This function utilizes the re module from Python's standard library to perform pattern matching using regular expressions. A fundamental requirement for a password to be considered strong is that its length must be a minimum of eight characters. The function strictly disallows the inclusion of any whitespace characters within the password string. It mandates that a strong password must contain at least one uppercase letter to enhance its complexity. Similarly, the password must also include at least one lowercase letter to meet the strength criteria. A strong password is required to incorporate at least one numerical digit within its sequence of characters. The presence of at least one special character, defined as any non-alphanumeric character, is another necessary condition for a strong password. The code includes a series of assert statements that serve as test cases to validate the correctness of the is\_strong\_password function's logic. Finally, after executing these test cases, a print statement confirms whether all the provided assertions have passed successfully, indicating the function behaves as expected for these specific inputs.

Task Description #2 (Number Classification with Loops – Apply AI for  
Edge Case Handling)  
• Task: Use AI to generate at least 3 assert test cases for a  
classify\_number(n) function. Implement using loops.  
• Requirements:  
o Classify numbers as Positive, Negative, or Zero.  
o Handle invalid inputs like strings and None.

o Include boundary conditions (-1, 0, 1).  
Example Assert Test Cases:  
assert classify\_number(10) == "Positive"  
assert classify\_number(-5) == "Negative"  
assert classify\_number(0) == "Zero"

**Prompt:-**

Task: Use AI to generate at least 3 assert test cases for a  
classify\_number(n) function. Implement using loops.  
• Requirements:  
o Classify numbers as Positive, Negative, or Zero.  
o Handle invalid inputs like strings and None.

o Include boundary conditions (-1, 0, 1).  
Example Assert Test Cases:  
assert classify\_number(10) == "Positive"  
assert classify\_number(-5) == "Negative"  
assert classify\_number(0) == "Zero"

Code And Output:-

A screenshot of a computer

AI-generated content may be incorrect.

**Code Explanation:-**

The code defines a function named classify\_number intended to categorize input values as positive, negative, or zero. The function includes initial checks to handle invalid inputs, specifically None or string types, returning "Invalid input" for these cases. It uses a for loop to check if the input number is equal to zero, returning "Zero" if a match is found. Another for loop is used to determine if the input number is positive, returning "Positive" if it's equal to 1 or greater than 0. A subsequent for loop checks if the input number is negative, returning "Negative" if it's equal to -1 or less than 0. If the input does not fall into the categories of zero, positive, or negative based on the defined loops, the function returns "Invalid input". The code includes several assert statements to test the classify\_number function with various numerical and non-numerical inputs. These assertion tests cover positive, negative, and zero values, as well as invalid inputs like strings and None. The purpose of these assertions is to verify that the function correctly classifies the numbers and handles invalid inputs as expected. A final print statement confirms that all the provided test cases have successfully passed, indicating the function works correctly for the given examples.

Task Description #3 (Anagram Checker – Apply AI for String Analysis)  
• Task: Use AI to generate at least 3 assert test cases for  
is\_anagram(str1, str2) and implement the function.  
• Requirements:  
o Ignore case, spaces, and punctuation.  
o Handle edge cases (empty strings, identical words).  
Example Assert Test Cases:  
assert is\_anagram("listen", "silent") == True  
assert is\_anagram("hello", "world") == False  
assert is\_anagram("Dormitory", "Dirty Room") == True.

**Prompt:-**

Task: Use AI to generate at least 3 assert test cases for  
is\_anagram(str1, str2) and implement the function.  
• Requirements:  
o Ignore case, spaces, and punctuation.  
o Handle edge cases (empty strings, identical words).  
Example Assert Test Cases:  
assert is\_anagram("listen", "silent") == True  
assert is\_anagram("hello", "world") == False  
assert is\_anagram("Dormitory", "Dirty Room") == True.

Code And Output:-

A screenshot of a computer

AI-generated content may be incorrect.

**Code Explanation:-**

The code defines a function called is\_anagram that takes two strings as input to determine if they are anagrams of each other. Inside the is\_anagram function, a helper function named clean is defined to preprocess the input strings. The clean helper function iterates through the input string, keeping only alphanumeric characters. It also converts all remaining characters to lowercase, effectively ignoring case and punctuation for the anagram check. The is\_anagram function applies this clean helper function to both input strings to obtain their cleaned versions. An edge case is handled where if both cleaned strings are empty, the function correctly identifies them as anagrams and returns True. For non-empty cleaned strings, the function sorts the characters of both cleaned strings alphabetically. It then compares the sorted strings to see if they are identical, which is the core logic for determining if they are anagrams. The code includes several assert statements to test the is\_anagram function with various examples, including anagrams, non-anagrams, and edge cases like empty strings and identical words. Finally, a print statement confirms that all the provided assertion tests have passed, indicating the function works as intended for these specific test cases.

Task Description #4 (Inventory Class – Apply AI to Simulate Real-  
World Inventory System)  
• Task: Ask AI to generate at least 3 assert-based tests for an  
Inventory class with stock management.  
• Methods:  
o add\_item(name, quantity)  
o remove\_item(name, quantity)  
o get\_stock(name)  
Example Assert Test Cases:  
inv = Inventory()  
inv.add\_item("Pen", 10)  
assert inv.get\_stock("Pen") == 10  
inv.remove\_item("Pen", 5)  
assert inv.get\_stock("Pen") == 5  
inv.add\_item("Book", 3)  
assert inv.get\_stock("Book") == 3

**Prompt:-**

• Task: Ask AI to generate at least 3 assert-based tests for an  
Inventory class with stock management.  
• Methods:  
o add\_item(name, quantity)  
o remove\_item(name, quantity)  
o get\_stock(name)  
Example Assert Test Cases:  
inv = Inventory()  
inv.add\_item("Pen", 10)  
assert inv.get\_stock("Pen") == 10  
inv.remove\_item("Pen", 5)  
assert inv.get\_stock("Pen") == 5  
inv.add\_item("Book", 3)  
assert inv.get\_stock("Book") == 3

Code And Output:-

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Code Explanation:-**

The code defines a Python class named Inventory to simulate a simple stock management system. The \_\_init\_\_ method initializes an empty dictionary called stock within each Inventory object to store item names and their quantities. The add\_item method allows adding a specified quantity of an item identified by its name to the inventory. If the item already exists in the inventory, its current stock is increased by the added quantity. If the item is not present, it is added to the stock dictionary with the given initial quantity. The remove\_item method is used to decrease the stock of an item by a specified quantity. This method ensures that the stock level for any item does not go below zero, preventing negative inventory counts. The get\_stock method provides a way to retrieve the current quantity of a specific item from the inventory. If the requested item is not found in the inventory's stock dictionary, the get\_stock method returns a default value of 0. The code includes several assert statements after the class definition to test the functionality of the add\_item, remove\_item, and get\_stock methods with various scenarios, verifying their correct behavior.

Task Description #5 (Date Validation & Formatting – Apply AI for  
Data Validation)  
• Task: Use AI to generate at least 3 assert test cases for  
validate\_and\_format\_date(date\_str) to check and convert dates.  
• Requirements:  
o Validate "MM/DD/YYYY" format.  
o Handle invalid dates.  
o Convert valid dates to "YYYY-MM-DD".  
Example Assert Test Cases:  
assert validate\_and\_format\_date("10/15/2023") == "2023-10-15"  
assert validate\_and\_format\_date("02/30/2023") == "Invalid Date"  
assert validate\_and\_format\_date("01/01/2024") == "2024-01-01"

**Prompt:-**

• Task: Use AI to generate at least 3 assert test cases for  
validate\_and\_format\_date(date\_str) to check and convert dates.  
• Requirements:  
o Validate "MM/DD/YYYY" format.  
o Handle invalid dates.  
o Convert valid dates to "YYYY-MM-DD".  
Example Assert Test Cases:  
assert validate\_and\_format\_date("10/15/2023") == "2023-10-15"  
assert validate\_and\_format\_date("02/30/2023") == "Invalid Date"  
assert validate\_and\_format\_date("01/01/2024") == "2024-01-01”

Code And Output:-

A screenshot of a computer

AI-generated content may be incorrect.

**Code Explanation:-**

The code imports the datetime class from Python's built-in datetime module to work with dates and times. It defines a function named validate\_and\_format\_date that takes a single argument, date\_str, which is expected to be a string representing a date. The purpose of this function is to attempt to parse the input string as a date and then reformat it into a different string format. The function specifically tries to parse the input date\_str assuming it is in the "MM/DD/YYYY" format using datetime.strptime. A try...except ValueError block is used to handle potential errors that may occur if the input string does not match the expected date format or represents an invalid date. If the parsing is successful within the try block, the parsed date object is then formatted into a new string using the "YYYY-MM-DD" format via strftime. This newly formatted date string in "YYYY-MM-DD" format is returned by the function upon successful validation and formatting. If a ValueError is caught during the parsing process, indicating an invalid or incorrectly formatted date string, the function returns the string "Invalid Date". The code includes multiple assert statements to test the validate\_and\_format\_date function with a variety of inputs, including valid dates in the expected format and invalid date strings. These assertion tests verify that the function correctly parses and formats valid dates while also correctly identifying and returning "Invalid Date" for invalid inputs.